

MAKING AN ADDING MACHINE

What's in Chapter 3

You have completed the Hello World applet and probably think programming in Java is very easy. This is one of the simplest Java applications you can develop other than the infamous “to do list” sample. Each chapter of this book increases in difficulty and complexity, and this chapter covers visual programming in more detail. You will build a logic bean and combine it with GUI beans to make an adding machine. You will write some simple Java code for the logic bean to perform the calculation function. This chapter covers the following topics:

JavaBeans basics

- GUI beans
- Nonvisual beans
- Composite beans
- Overview of Layout Managers

Building an Adding Machine

- Using a GridLayout
- Using TextFields
- Setting the tab order of GUI beans
- Adding the contents of two TextFields

As you go through this book, we cover some object-oriented and visual development concepts. Many books exhaustively cover a particular object-oriented methodology or specific design technique. Because this book focuses on implementation, it covers only those object-oriented design concepts necessary to understand the applications you are building. The Adding Machine applet uses the Model-View-Controller design pattern that is common in object-oriented programming. First, you build the view, then you build the model part. This is okay, but in real life you will usually develop the model parts first; then you develop the view parts, because view (or GUI) parts should not influence the implementation of model parts. It is unrealistic to say that view parts have no influence on model parts, but this influence should be minimal.

Now that you have completed a simple HelloWorld applet, it's time to look back at the steps and understand how you built it. Before you go on to more

complex programs, you need to understand the different types of JavaBeans. This will help you design and implement your own Java applications. The JavaBeans specification is a fundamental concept of JDK Version 1.1, and you need to understand JavaBeans to fully utilize VisualAge for Java.

JavaBeans Basics

The notion of JavaBeans was introduced in JDK 1.1; VisualAge for Java uses JavaBeans and generates JavaBeans. End users of a program using JavaBeans neither know nor have to care that JavaBeans were used. JavaBeans were introduced primarily to assist Java developers; there are many changes to the JDK to support JavaBeans. In this section, you will learn the basics behind JavaBeans, and in later chapters, you will learn more detailed information about JavaBeans classes and methods. JavaBeans are used throughout this book and in Chapter 14, “Servlets” and Chapter 15, “Using Enterprise JavaBeans,” you learn how to use JavaBeans in server-side programming. JDK 1.2 builds on JavaBeans basics and adds methods and new classes, but many of the package names are different.

NOTE

There is a separate version of VisualAge for Java that supports JDK 1.2 that you need to install if you want to use these new Java features. If you move Java code from JDK 1.1 to JDK 1.2, you will also need to modify the code to reflect the new package names.

Different types of beans or classes are used in the Visual Composition Editor (VCE). There are *GUI* beans, *logic* beans (also called invisible or nonvisual beans), and *composite* beans. This section explains the three types of beans. These JavaBeans terms are used by all Java different tools that support JavaBeans, and they are different from the traditional VisualAge terms that have been used for years. If you are a veteran VisualAge developer, you will recognize that GUI beans are the same as Visual Parts, and logic or invisible beans are the same as Nonvisual Parts.

What Are JavaBeans?

GUI beans and logic beans are Java classes, but there is a bit more to it than that. A comprehensive JavaBeans specification that covers all requirements for a JavaBeans component is available from the Sun web site at www.javasoft.com. There are two key requirements for JavaBeans. First they must have a default constructor that takes no parameters. Classes can have many constructors, but the visual builder tools rely on the default constructor to instantiate the class.

Secondly JavaBeans need to be serializable, which means the class declaration needs to implement **Serializable** and implement the appropriate methods. VisualAge for Java does not require that JavaBeans implement serializable as most other Java development IDEs. In Chapter 8, “Deploying Java,” you will see how to use nondefault constructors in the VCE.

NOTE

Default constructors can call nondefault constructors, ones that require parameters. You can do this in the constructor code by hard coding default values for the parameters. This may not seem like a good idea, but remember that you just need a JavaBeans instance at development time and you can change properties later.

A few Java books cover the bean specification in great depth, and there are a ton of articles that give various varying overviews of JavaBeans. If you are a Java tool developer, you need to be acquainted with what is needed for JavaBeans. Studying at least one in-depth book on the bean specification is helpful but not required, but it would not hurt. Fortunately, VisualAge for Java generates JavaBeans and therefore insulates you from some of the technical details relating to JavaBeans. However, it is important for you to understand how JavaBeans work.

The entire JDK class library was updated to support JavaBeans, including the new V1.1 event model. You can see how the event model works by examining the code generated by VisualAge for Java. Because the JDK classes implement new methods supporting the new event model, all AWT-based classes inherit these special functions, which support the notification or messaging framework in the JDK class library. Implementation of a notification framework enables beans to notify or send messages to other beans. This critical function is what enables you to build program elements by making connections. As stated previously, this changes the way you develop your Java applications (it is better object orientation), but the end user will not notice any difference.

Deprecated Methods

Java 1.0 classes will work with Java 1.1 run-time support, but the Java 1.0 classes do not use the methods implementing the different event model in JDK 1.1. Although many Java V1.0 classes are beans, they are stale beans that need some refining to be fully compatible with other JavaBeans. A number of methods in the JDK class library were replaced by new methods, and these obsolete methods are referred to as *deprecated methods*. An example of a deprecated method is the

Button **enable()** method, which is replaced with the `setEnabled(true)` method. These old methods work in the compiler and at run time in the virtual machine, but eventually, they will not be supported. It is a good idea to use the new JDK methods.

Types of Beans

The most atomic or granular bean is called a *primitive* bean. GUI and invisible beans can be primitive beans. Examples of primitive beans from the Java class library are `Button`, which is a GUI bean, and `Color`, which is an invisible bean. You can change the settings and default properties of primitive beans in the VCE.

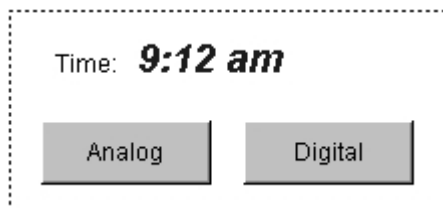
Beans can be combined to create *composite* beans. When you combine two or more logic beans, you have a composite logic bean. For example, if you are building the logic component of a `Clock` bean, you could combine the `Timer` and `Date` logic beans in a composite logic bean named `Clock`. The `Clock` bean would supply the services you would expect from such a bean, such as setting and getting the time and date.

You create a composite GUI bean when you combine a GUI bean with one or more other beans. For example, you can combine the previously discussed logic `Clock` bean with a user interface bean that displays the time and date and has buttons to perform the various clock functions. You could call this bean a `ClockView` bean as shown in Figure 3.1. In fact, the clock could have multiple views like digital, analog, or a combination, where the date is shown in digital format and the time as a traditional clock with hands ticking.

GUI Beans

GUI beans are user interface controls in the AWT or JFC class library, such as `Button`, `TextField`, and `JFrame`. The VCE generates Java code for all of the GUI beans. GUI beans supplied with the VCE are sometimes called *controls* or *widgets*. The terms *controls* and *widgets* came about because the higher level definitions in the Java AWT essentially map to the underlying graphical API for each windowing system running Java; whereas the JFC controls do not directly map to the operating system graphical controls. When talking about GUI beans in this book, we will refer to them as beans or controls interchangeably.

Figure 3.1 `ClockView` composite GUI part.



All GUI beans are subclasses of **Component**, which is an abstract base class in the Java class library. The **Component** class sets the base behavior for all user interface controls and for **Container**, too. You can see this inheritance relationship in Figure 3.2.

Finding Beans

You may want to search for other JavaBeans in the Workspace when you are developing Java programs. Let's try searching for *Applet* with the following steps:



On the Workbench toolbar, select **Search the** button.

The Search dialog appears, as shown in Figure 3.3, allowing you to enter the following:

Enter **Applet** in the Search String field.

Select **Workspace** radio button for the Scope.

Select **Declarations** radio button for the Usage.

Press the **Start** button to begin the search.

If there are no matches to the search string, the short message **No matches found** appears at the bottom of the Search dialog. It's a bit subtle, so be careful because you may think VisualAge for Java is still searching. Also take care not to waste time searching the entire Workspace for a common class reference. You can always click the **Stop** button to end a search that is out of control. Once the Search Results window opens, as shown in Figure 3.4, you can view the declaration and methods in the Java Applet class.

You will often search for a method or a class that is referenced in many places. When the search returns a list, you usually open a view or browser to inspect the class in detail. Open a browser for the Applet by selecting the Applet in the upper left pane, and from its pop-up menu select the **Open** menu item.

When the Class browser opens, select the Hierarchy tab on the view as shown in Figure 3.5.

Figure 3.2 Button inheritance hierarchy.

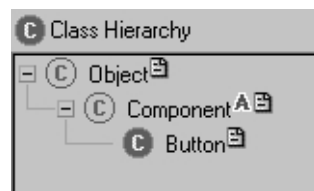
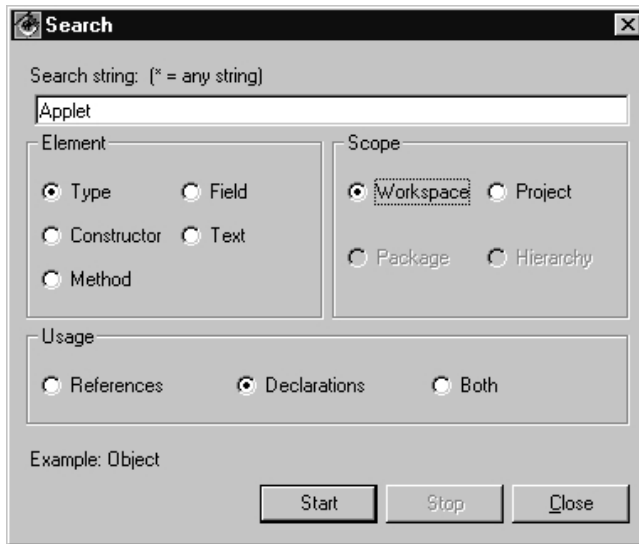


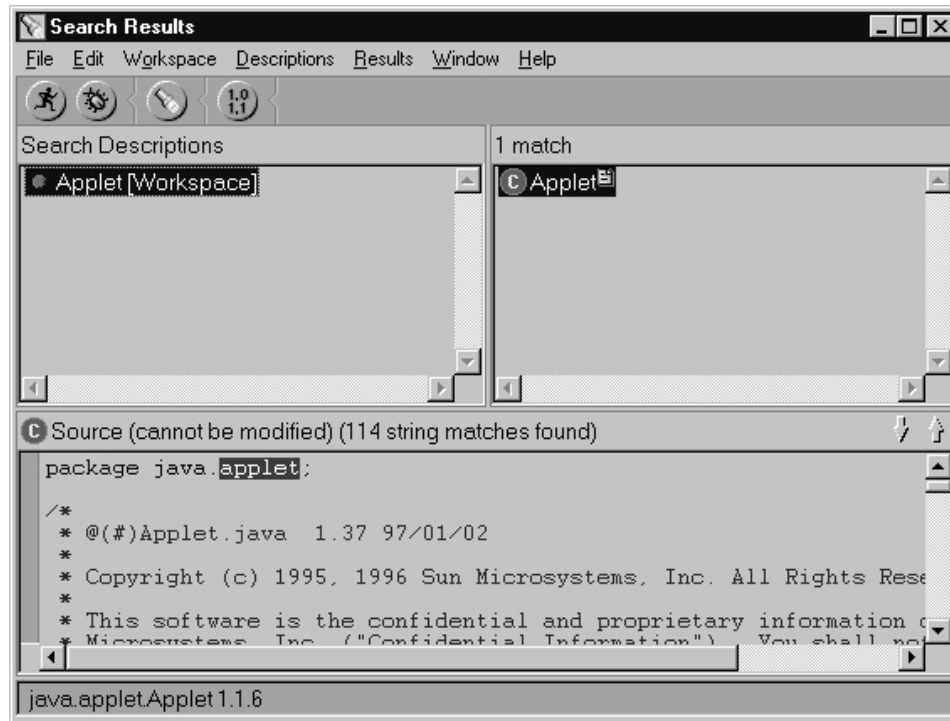
Figure 3.3 Search dialog.**Tip**

Be careful when searching. You should try to scope the search as much as possible to save time. It is a good idea to look for declarations. Also, there are usually many references in the Workspace, and you can scope the search to the Project or Package to speed things up.

Now that you have found the Applet class, you can browse its definition and learn more about the class and its superclasses. If you ever need information about a bean, the search feature in the IDE is a fast way to get to that information. It will also verify whether the class is loaded in the Workspace and available to use.

Tip

The Search Results window is non-modal, which means that you can work on other views or browsers while the Search Results window is open. You can also run multiple searches and have multiple Search Results windows.

Figure 3.4 Search results.

Scoping a Search

VisualAge for Java Version 3 has a new search feature called a *Working Set*. When you are working on certain projects or packages in the IDE, you can define a Working Set that uses these project or packages for a search. Searching a Working Set is much faster than searching the entire Workbench for a class definition. Let's set up a Working Set with the following steps:

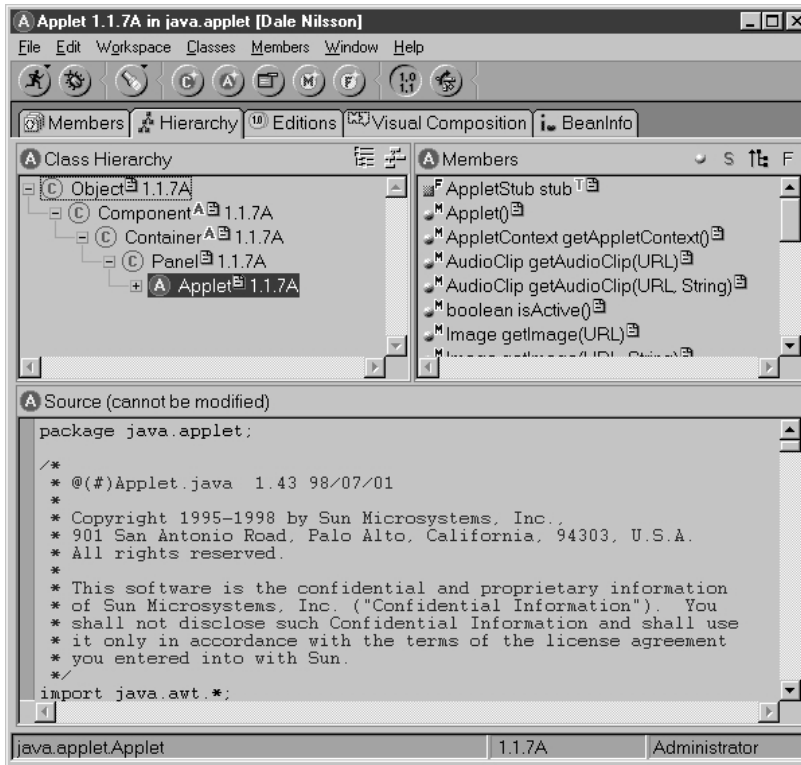


Press the Search button and the Search Dialog displays as shown in Figure 3.6.

Select the **Working Set** radio button, and press **Choose**, and the Choose dialog displays.

Select the **New** button on the Choose dialog and another dialog displays.

Enter **Hanoi** for the Working Set name.

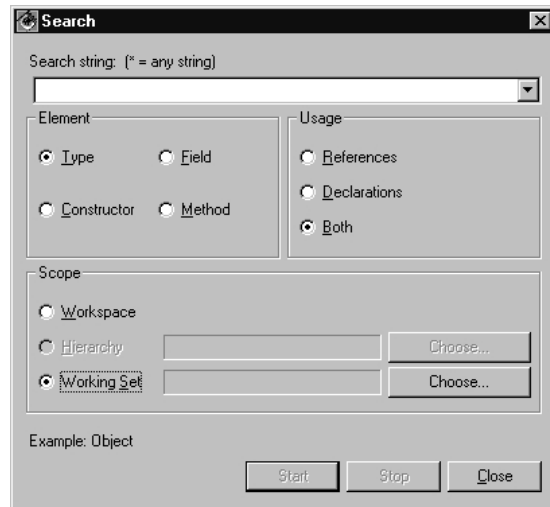
Figure 3.5 Applet hierarchy.

Select the `com.ibm.ivj.examples.hanoi` package in the tree view on the left side of the window as shown in Figure 3.7.

As you can see from Figure 3.7, a Working Set can have a number of projects or packages. You can now use the Hanoi Working Set to easily limit searches to that package. The Working Set definition is saved with the Workspace and can be revised or deleted in the future. When you are finished, close the Search dialog.

Logic Beans

Logic or nonvisual beans contain business logic, such as mathematical computation, data access functions, and application logic. The JDK class libraries come with a number of logic beans that are very helpful in building applications. Most of these beans are designed for general-purpose use, and in many cases, they need to be subclassed to add application-specific function. Figure 3.8 shows the composition of the ClockView application, combining both GUI and logic beans.

Figure 3.6 Search Working Set.

Logic JavaBeans are part of good object oriented design because they allow the use of Model-View separation. This allows you to isolate or encapsulate the logic from its view or visual representation. The logic beans can be reused with other views and can be moved to a server.

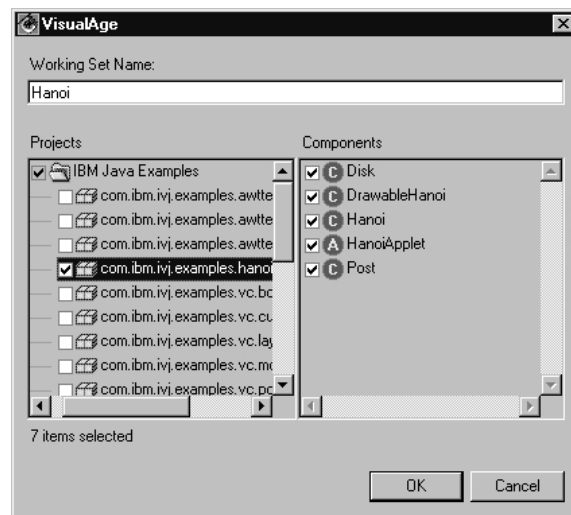
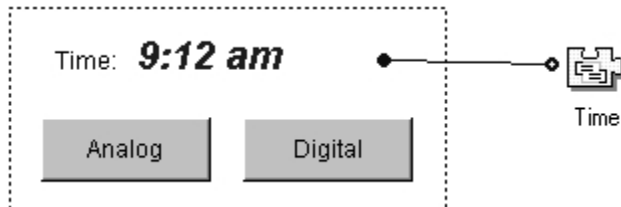
Figure 3.7 New Working Set.

Figure 3.8 ClockView composite with logic beans.

Building an Adding Machine

In this chapter, you will build an Adding Machine application that combines GUI and logic beans. First, you will construct the GUI bean, `CalculatorView`, and then you will develop a logic bean, `Calc`, that you will use in combination with the GUI bean to perform the calculations.

The Adding Machine enables you to enter two numbers, press a button to add the contents of the two `TextFields`, and display the result in the output field. The application window has:

- Three `TextFields`, two for accepting the numbers to be added and one to display the result

- Three Labels for the `TextFields`

- A Button for the Add function

- A logic bean, which performs the addition and signals that the operation has been performed and that the result property in the `Calc` bean has changed

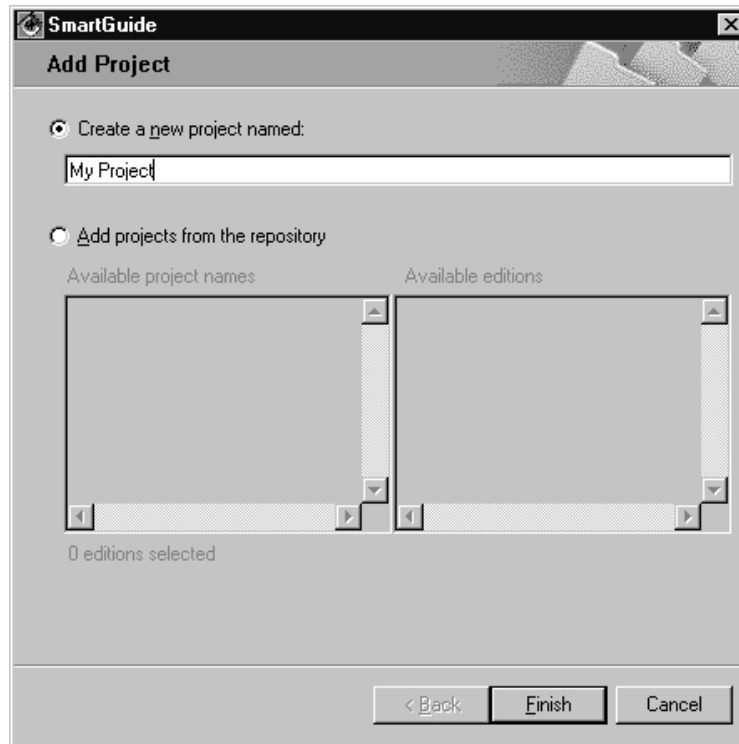
Let's begin by making a new applet in the Visual Composition Editor to build the view of the Adding Machine. You will build and test the `CalculatorView` GUI bean before building the `Calc` logic bean. This is a new application, so you should make a new package before starting the new class. By creating the new package, you keep the classes, methods, and ultimately the files for the different applications separate. If the VisualAge for Java Workbench is not started, restart it now.

Creating a New Package

First, create the new package for this program in the project used for the Hello World applet.

- In the Workbench, move to the upper pane and select the **My Project** project.

- If you skipped Chapter 2, "Building the Hello World Applet," you need to create a new project in the Workbench. From the menu bar, select

Figure 3.9 Add Project SmartGuide.

Selected => Add => Project, or else select the **Project** icon on the tool bar. This displays the project SmartGuide, as shown in Figure 3.9, where you can enter the project name. Once you have a project to add new classes, you can proceed.

The next thing to do is to create the package, which will contain all the components of the Simple Adder applet.

With **My Project** selected, right-click on the top pane, and from the pop-up menu select **Add Package**.

In the **Create a new package named** entry field, enter **calculator** and click. Press the **Finish** button to create the package, as shown in Figure 3.10.

After the calculator package is created, your Workbench should look like Figure 3.11.

Creating a New Class

Now that you have defined the project and the package in the IDE, you can start defining the beans that will make up the Simple Adder. The project and package

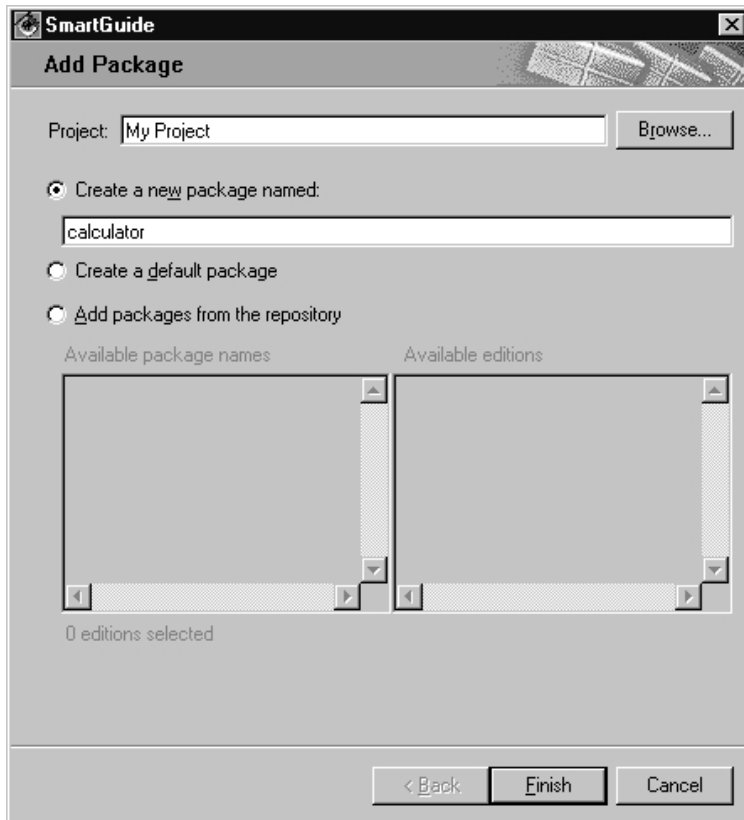
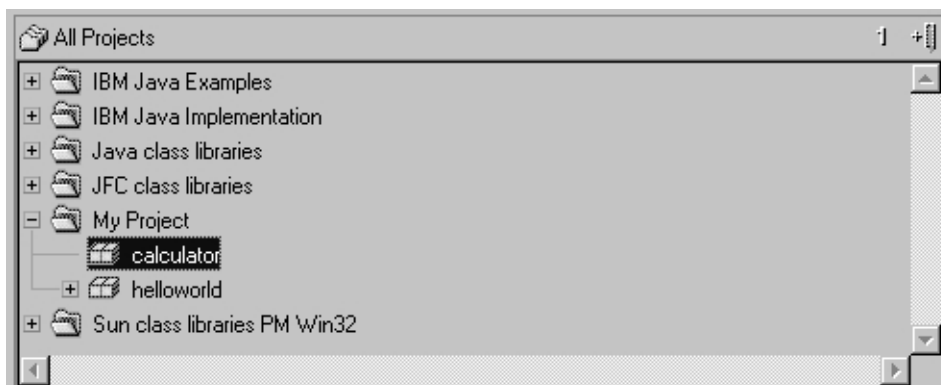
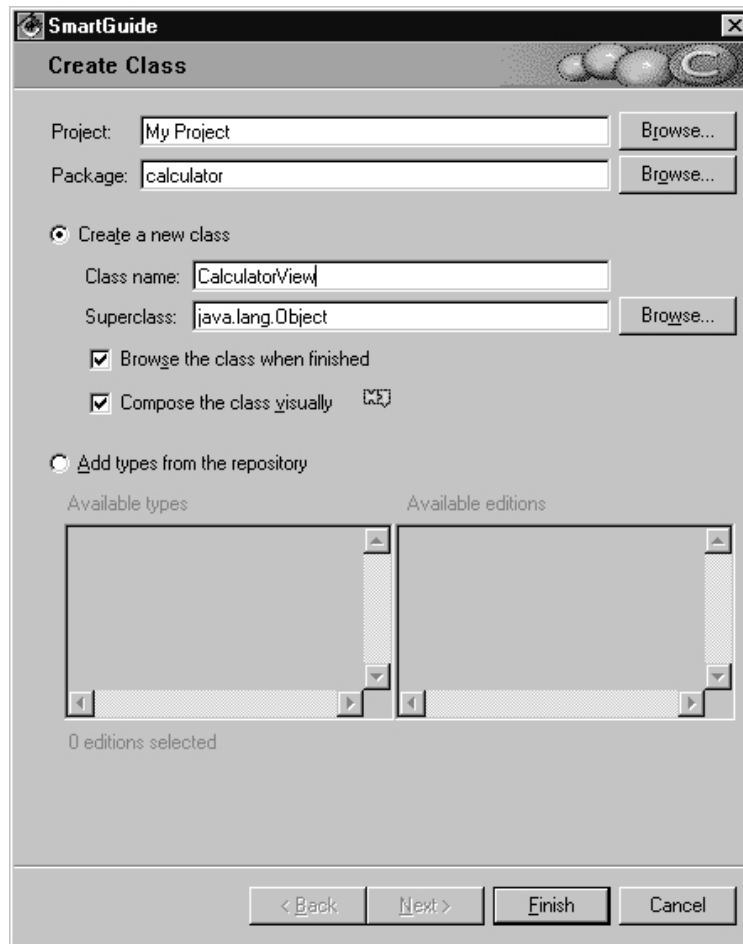
Figure 3.10 Add Package SmartGuide.**Figure 3.11 Workbench with My Project and calculator package.**

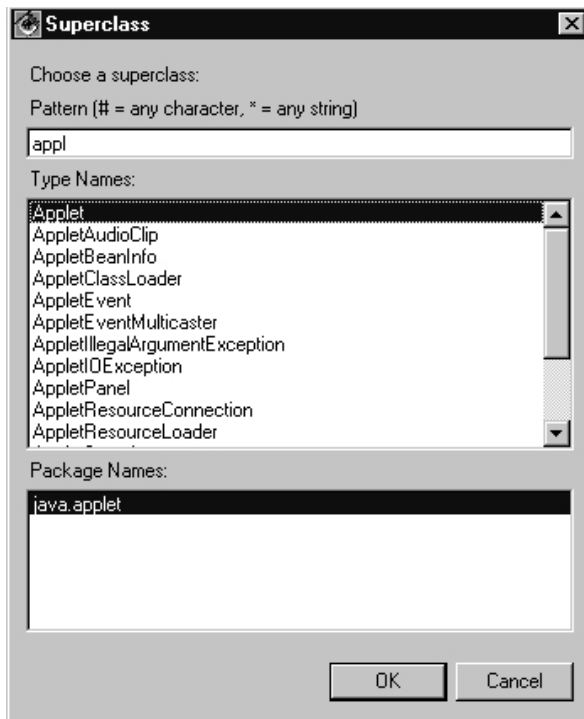
Figure 3.12 Create Class SmartGuide.

provide the structure for the IDE to contain and catalog your beans in the Workspace. Packages are also used when exporting your Java classes. To begin building the CalculatorView with the following steps:

With the calculator package selected, right-click the top pane, and from the pop-up menu select **Add => Class**.

In the Create Class SmartGuide shown in Figure 3.12, enter **CalculatorView** in the **Class name** entry field.

Make sure you capitalize the class name. It is a Java convention to capitalize class or bean names, and this is for a very good reason. Beans have constructors

Figure 3.13 Class Qualification dialog.

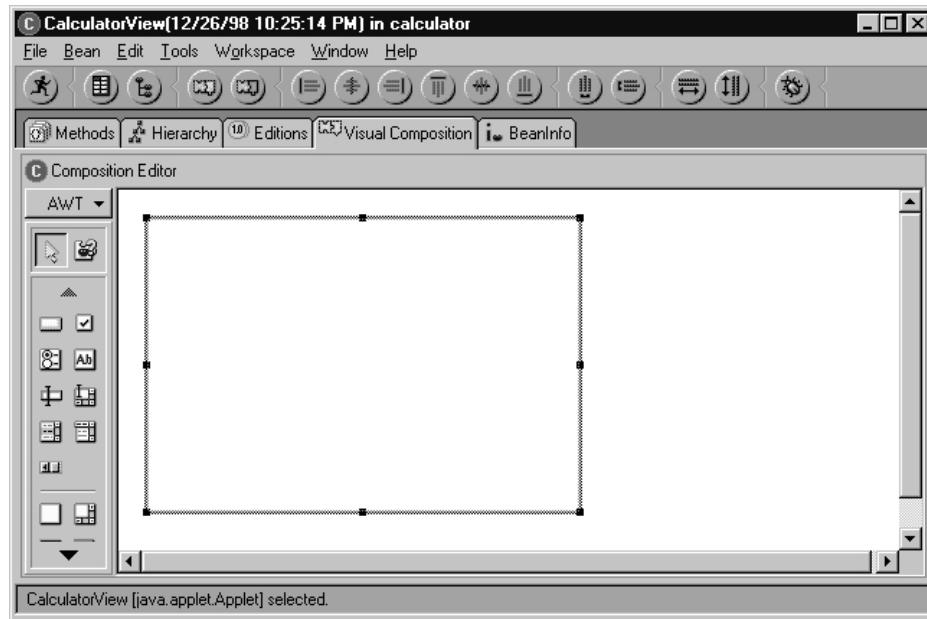
that are essentially methods with no return value. Capitalizing bean names makes it easy to identify constructors, because they are capitalized methods with the same name as the bean.

Press the **Browse** button opposite the **Superclass** entry field, and type the word **Applet** to select the Applet class from the java.applet package. Notice that as you type in the SuperClass Dialog, the choice of class names is narrowed down as shown in Figure 3.13. You may need to use the scroll bar in the *Type Names:* list to see the Applet class listed at the top.

Once the Applet bean is selected, press the **OK** button to save this selection, and close the Class Qualifications Dialog.

Make sure that the **Design the class visually** radio button is selected on the Smart Guide, as shown in Figure 3.12. This radio button is merely a convenience; it causes the browser to automatically open to the VCE.

Press the **Finish** button to complete the specifications for the applet so VisualAge for Java can generate the code to create the **CalculatorView**

Figure 3.14 Visual Composition Editor.

class. You can use the standard class `SmartGuide` for simple applets. Later you will use this same `SmartGuide` to make logic `JavaBeans`.

The VCE now opens, enabling you to place the visual components that make up the `CalculatorView`, as shown in Figure 3.14. The browser for the `CalculatorView` class automatically opens to the VCE page because you told the `SmartGuide` that you wanted to design this class visually. This is primarily a convenience—you can simply open a visual class in the Workbench and flip to the VCE page at any time. As you develop Java programs using the VCE in this book, many of the screen captures in the VCE will show the design surface and omit the toolbar and other areas when they aren't used.

Layout Managers

The `HelloWorld` applet was built using no `Layout Manager`; actually, it had a `Layout Manager` set to `null`. Using a null layout is not a good design, because it uses pixel coordinates to position the controls. This creates a new variation for Java applications: write once and run anywhere, but it will look different with different browsers and various screen resolutions. The net effect is that a perfectly aligned applet on your screen may appear jumbled and ugly elsewhere. Java

Layout Managers are provided to handle the run-time alignment and positioning of GUI JavaBeans. Layout Managers are properties of Java **Containers**, so the Layout Manager can be easily set in the property sheet.

Tip

There is a real cool stealth feature in VisualAge for Java Version 3.0 that transforms a null Layout to a GridBagLayout automatically. You will see how this feature works in Chapter 6, “Building the Advanced Calculator GUI.” This feature provides one way to use a good GridBag layout, but you still should become familiar with the different layout managers and their behavior.

Containers can use one of the Layout Managers in the JDK or you can also make your own Layout Managers. This means that Frames, Applets, and Panels can all have their own Layout Managers. In this chapter, you will use GridLayout; later, when you build the Advanced Calculator, you will use multiple panels with different Layout Managers. When you combine Containers with different Layout Managers, you must be careful, because the different Layout Managers can produce conflicting behavior that will not provide the result you expect. This program is an Applet, and because an Applet is a Container, it will have its own Layout Manager.

Layout Managers are used to align, space, and size controls placed on an Applet, Panel or other Container. There are several types of Layout Managers, and only experience will make you comfortable using them. Following are some of the basic Layout Managers in the JDK:

- BorderLayout
- BoxLayout
- CardLayout
- FlowLayout
- GridBagLayout
- GridLayout

Each Layout Manager provides its own behavior and can be customized within the VCE. Each standard Layout Manager subclasses Object and implements either the LayoutManager or LayoutManager2 interface. Many Containers have a default Layout Manager if no Layout Manager is specified. It is best to specify the Layout Manager to insure your Container will perform as you expect. There is also a layout called *null*, which forces the positions of GUI beans to be fixed in the Container. In later chapters, you will use other Layout Managers like the Border, Flow, and GridBag layouts.

Custom Layouts

The VCE lists all the Java Layout Managers in a Container's property sheet. Version 3 includes all user-defined Layout Managers loaded in the Workspace. If you create your own Layout Manager, it can now be used in the VCE.

The VCE always generates a `setLayout()` method in the `init()` method of an Applet. The code for the `init()` method is shown in the next section. You would enter the code for the `setLayout()` method in a user code area provided in the generated code. All of the other code is regenerated, so you should put code here at your own risk. You can see the warning in the code that this method will be regenerated, and you will lose changes to the source code.

You can change the comment heading for regenerated methods. This can be found in the Workbench Options dialog. You can also add comments to the user code areas. The following code for the `CalculatorView` `init()` method, viewable from the Methods page, shows some user exits for Java code:

```
/**
 * Handle the Applet init method.
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
public void init() {
    super.init();
    try {
        setName("CalculatorView");
        setLayout(null);
        setSize(300, 203);
        // user code begin {1}
        // user code end
    } catch (java.lang.Throwable ivjExc) {
        // user code begin {2}
        // user code end
        handleException(ivjExc);
    }
}
```

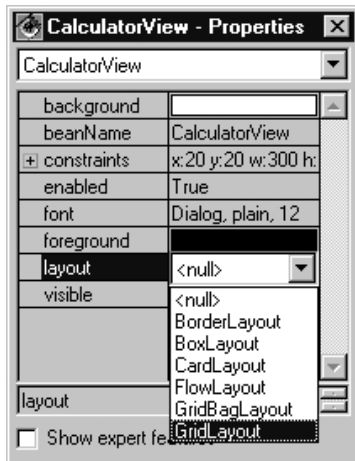
Using a GridLayout

For the Adder applet, you will use `GridLayout`, which is used when you want to align a number of same-sized beans. This layout is frequently used for entry forms, because it can keep `TextFields` and `Labels` aligned.

Place the mouse over the applet and open its property sheet.

Select the **layout** field, which then displays a button.

Select the Layout Manager drop-down list, and select `GridLayout` from the list as shown in Figure 3.15.

Figure 3.15 Layout properties.

Now that you have changed the layout to `GridLayout`, some properties for the `GridLayout` are displayed in the property editor. The Visual Composition Editor assumes 1 row and 0 columns. You will usually need to change these properties to achieve the results that you want instead of what the default `GridLayout` behavior. Use the following steps to set properties for `GridLayout`:

- Expand the layout **+** in the property editor.

- Set the **columns** to 1.

- Set the **rows** to 7.

Close the property sheet to save these changes. The Applet will not look any different, because the Layout Manager will only affect the GUI beans placed on it, and currently the Applet has no GUI beans.

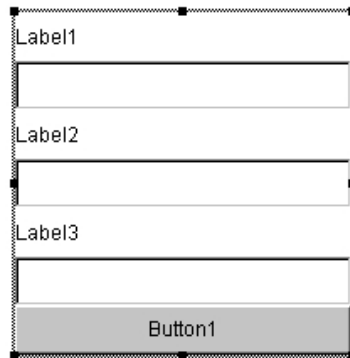
Adding GUI Beans to a GridLayout

The Simple Adder needs the GUI beans for the user interface. Add the needed GUI beans to the `GridLayout` with the following steps:

- First, make the Applet smaller by resizing it. Select the Applet with the left mouse button, select one of the handles (the black squares in the corners), and drag the handle to resize the Applet.



- Place three **Label** beans alternating with three **TextField** controls on the `GridLayout` on the applet. See Figure 3.16 for suggested placing of controls. You may need to drag and drop some of the controls until you like the Applet's appearance. The ability to dynamically reposition GUI JavaBeans is very helpful in designing user interfaces.

Figure 3.16 Applet with GUI beans.**Tip**

VisualAge for Java Version 1 had a **Sticky** checkbox to make it easier to drop multiple controls of the same type. Version 3 does not have the Sticky checkbox—you need to press the Ctrl key when you select the icon in the VCE palette. You can also use the clipboard copy/paste features to copy several of the same controls. You can even use copy/paste to copy JavaBeans between different VCE windows.



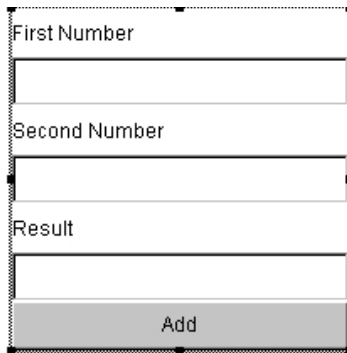
Select a **Button**, move the mouse pointer to a position near the bottom of the dashed box on the VCE, and press the left mouse button to drop the Button.

Setting the Text Property

Many GUI JavaBeans have a *text* property that is a Java String that holds the value of the characters displayed on the GUI bean. But Buttons are special; they have a *label* property for the same purpose. You need to edit these properties in the CalculatorView Applet with the following:

Open the property for the Button and change the *label* property of the Button to **Add**.

Edit the *text* properties for the Label controls to describe the TextFields below them. Use **First number**, **Second number**, **Result** for the respective Labels. When you finish the CalculatorView, it should look like Figure 3.17. You can also edit the Label text in the property sheet.

Figure 3.17 Beans placement.

The image shows a Java Swing window titled "Beans placement". Inside the window, there are three text fields and one button. The first text field is labeled "First Number", the second is labeled "Second Number", and the third is labeled "Result". The button is labeled "Add". The text fields are arranged vertically, and the button is at the bottom.

Naming Beans

Each bean you drop has a default name. The first TextField control you drop is called TextField1. In addition, the first TextField you drop on the next composite you build will also be named TextField1. You can see that the proliferation of beans named TextField1 could cause a debugging nightmare. Changing the name to something more meaningful will help you follow the generated code, and it will provide some level of documentation for the code. So give the beans appropriate names. Change the bean names with these steps:

- Point to a GUI bean, click the right mouse button, and select **Change Bean Name** from the pop-up menu.

- Change the names of the TextFields to **tfNum1**, **tfNum2**, and **tfResult**.

- Change the name of the Button to **pbAdd**.

Read-Only TextFields

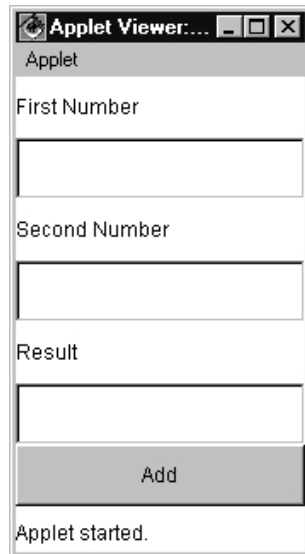
Because the result of the calculation will be supplied by a logic bean that is yet to be built, the result TextField should not allow keyboard input. One way to achieve this is to set the *editable* property to **False**.

- Double-click on the tfResult TextField to open its property sheet.

- Select the **editable => field;** from its pull-down menu, select **False**.

- Close the property sheet to save the change.

If you make any changes to a control in a property sheet and decide you do not like the changes, there is an easy way to go back to the old settings. After you close the property sheet, you can go to the **Edit** menu of the Visual Composition Editor and select **Undo**. You can step back through the changes until you get the

Figure 3.18 CalculatorView applet running.

bean back to the way you want it. Remember that **Edit/Undo** is your friend. There is also a **Redo** function if you go too far in your undoing.

Testing an Applet



Now you are ready to run this applet for the first time. From the upper tool bar of the VCE, press the **Run** button. This generates run-time code for the applet.

Once compilation ends, the appletviewer window appears using the default size and parameters for the applet. The appletviewer starts and runs the CalculatorView applet, as shown in Figure 3.18.

You can move between the first two fields, enter values, and press the **Add** button (of course, nothing happens yet). Nothing can be entered in the **Result** field because you made it noneditable. If you use the mouse to increase or decrease the window size, the GUI beans stay centered in the applet and the GUI beans expand, thanks to the GridLayout. It is usually good to have TextFields expand, but Buttons should not expand in the user interface. Close the applet viewer when you finish testing it.

Naming Conventions

Before you begin building the application is a good time to have a short discussion of naming conventions. When developing applications, it is prudent to adopt some consistent naming conventions. Java is a case-sensitive, type-specific

language, which means that each class, method, and data field must have a unique name and a defined type. Use descriptive names to make your beans more usable. A good guideline is to have a set of prefixes and suffixes that can ensure consistent naming. For example, the Hello World application used the *pb* prefix for Buttons. Most people have their own naming conventions as well as coding style conventions. In this book, the names of class instance variables start with a lowercase character.

Each word in the name of a class or bean also starts with an uppercase character. For example, the adding machine view is called `CalculatorView`. Any properties of a class or bean start with a lowercase character. Each subsequent word in the name of the property starts in uppercase. For example, the property that holds the result of the calculation is called *result*. The convention for instances of a class is the same as for properties. For example, the instance of the entry field that displays the calculation result is called *tfResult*.

When you create an instance of a class, you should try to identify the type of class you are instantiating. For example, you can use *tf* as the beginning of the instance name for an entry field. Adhering to these simple suggestions makes it easier to follow and understand the generated code, as well as to understand the connection messages that are issued by the program at run time. It also improves debugging, in the rare event that you make a programming error.

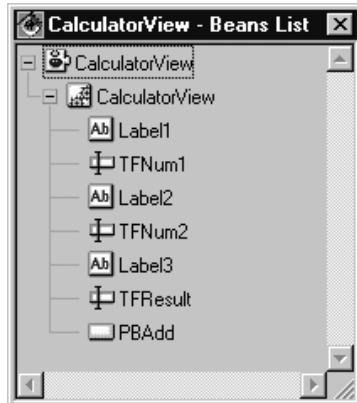
What Are Tab Stops?

Tabbing is a useful feature in user interfaces that improves screen navigation. The term *Tab stop* describes the position or location where the cursor goes when you press the Tab key. Each control placed on a canvas can be designated as a Tab stop. This designation determines the cursor position after the user presses the arrow or Tab keys.

Pressing the Tab key moves the cursor from its current position to the next control that has been designated a tab stop. The tabbing sequence is determined by the order in which the controls occur in the generated Java code. This order is determined by the order that controls are placed in the container. This order can be easily viewed and changed visually.

Setting the Tab Order

This section describes how you can define Tab stops and tabbing order. By default, AWT GUI beans have tabbing function automatically at run time. Tabbing order can be determined from the Applet's pop-up menu. To see an Applet's current tabbing order, select the Applet. Make sure the tip of the mouse pointer is not over one of the GUI beans; it should be at the bottom of the Applet in the small gap between the Button and the Applet. If the container is completely covered by GUI components, you will need to use the Beans List described in the next section.

Figure 3.19 Beans List for the applet.

Using the Beans List

As you can see, selecting some beans like Containers can be difficult. Sometimes the GUI beans are hidden or covered by other beans. In the case of Containers, controls can fill the Container bean so you can't access it. A special window enables you to select and work on beans. The Beans List shows all the JavaBeans in the VCE and all the visual connections that are listed at the bottom. Let's use the Beans List window to view the Tab Stops with the following steps:



From the menu bar, select **Tools => Beans List**. The Beans List window appears. You could also use the **Beans List** icon on the tool bar to open this window.

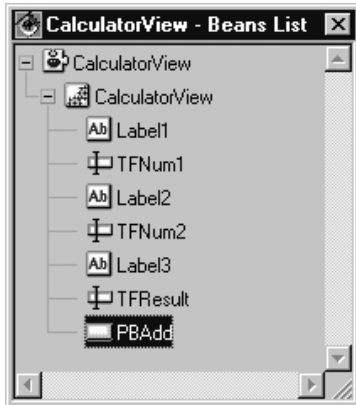
Expand the CalculatorView by pressing the expansion (+) icon as shown in Figure 3.19.

This window is very helpful for accessing beans in a Container; it provides a graphical tree view of all beans associated with this composite bean. This window lists all beans in this Applet, so you can easily select a bean and access all functions on its pop-up menu, like Properties, Delete, Open, and also the visual connections at the bottom. You can also drag and drop components in this window to put them in the desired order. The VCE will then generate the code with the components in the new order. If the TextFields and Buttons do not appear in the proper order, you can drag and drop them into the proper sequence by:

Selecting a control.

Using the mouse to drag the control to the proper position.

Releasing the mouse button to drop the control and set its position.

Figure 3.20 Setting Tab stops.

Now you are ready to see the tab stops for the Applet by following these steps:

- Select the second **CalculatorView** item and press the right mouse button to get the pop-up menu.

- Select **Set tabbing => Show Tab Tags** to see the tabbing order as shown in Figure 3.20.

Tip

After you have finished adjusting the Tab tags, you can hide them using the same process. When the Tab tags are showing, the pop-up menu item for the applet shows **Hide Tab Tags**.

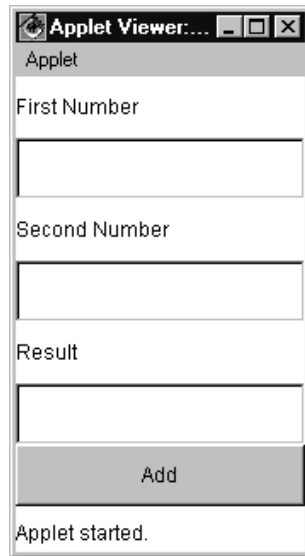
The VCE now shows the Applet with little yellow tab tags on each control in the Applet. The number in the tab tag indicates the sequence that the cursor will follow when you press the Tab key. If the TextFields and Buttons do not appear in the proper order, you can drag and drop the tab tags in the proper sequence. Set the tabbing order so the tabbing sequence starts with **tfNum1** and then goes to **tfNum2**, **tfResult**, and **pbAdd**, as shown in Figure 3.19, with these steps:

- Select one of the tags.

- Drag the tag to the proper position.

- Drop the tag to set its position.

- Continue dragging and dropping the tab tags until they are in the proper order.

Figure 3.21 Running the Adding Machine.

The new tabbing order will be saved when you save the applet. The tab tags show only at development time, not at run time. You can hide the tags with these steps:

From the Beans List, select the pop-up menu for CalculatorView.

Select **Set Tabbing => Hide Tab Tags**.

Close the Beans List window because it is no longer needed.

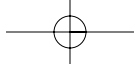
Running the Updated Adding Machine

You have completed the user interface for the CalculatorView applet. Now it is time to save and run the updated applet.



Select the Run tool bar button to save, generate, compile, and run the applet.

When the applet starts, it should look like Figure 3.21. The user interface looks like it did the last time you ran the Adding Machine. You can enter numbers in the TextFields, and the tabbing works correctly. When you are finished reviewing the running applet, you can leave the applet running. In the next chapter you will add the logic to the applet, and you will be able to use that function by merely reloading the applet.



Summary

This was a little tougher than the Hello World application, but it was still pretty easy. You still need to make a logic bean with the add function, which is covered in the next chapter. In this chapter you learned the following:

VisualAge JavaBeans Basics including:

- Composite beans
- GUI beans
- Logic beans

How to Search in the IDE and define a Working Set

How to build the Adding Machine application user interface that included:

- Using a GridLayout
- Adding TextField, Label and Button beans
- Setting Tab order

These are the very basics of GUI design programming. In the following chapter, you will create a logic bean with the add function, complete the CalculatorView, learn another important aspect to JavaBeans; namely bean features, and define JavaBeans properties and methods.

